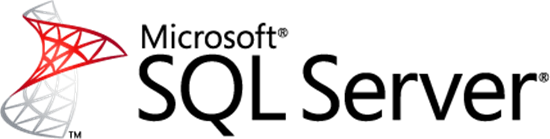
**New Spatial Features in SQL Server Code-Named “Denali”**

http://social.technet.microsoft.com/wiki/Utility/images/star-left-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-right-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-left-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-right-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-left-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-right-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-left-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-right-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-left-on.pnghttp://social.technet.microsoft.com/wiki/Utility/images/star-right-on.png

[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/0523.SQL_5F00_h_5F00_rgb.png)

SQL Server Code-Named “Denali” introduces several significant enhancements to spatial types, such as support for new subtypes of circular arcs, new and updated methods and aggregates for all subtypes, improved precision, and updates to the geography type.   
  
**Authors:** Ed Katibah (Microsoft) and Milan Stojic (Microsoft)

**Technical Reviewers:** Marko Tintor (Microsoft), Michael Kallay (Microsoft), Michael Rys (Microsoft) and Bob Beauchemin (SQLSkills)  
  
***Note****: you may download this document as a Microsoft Word file from* [*sqlcat.com* [http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-components-sitefiles/10_5F00_external.png](http://sqlcat.com/whitepapers/archive/2011/08/08/new-spatial-features-in-sql-server-code-named-denali-community-technology-preview-3.aspx)](http://sqlcat.com/whitepapers/archive/2011/08/08/new-spatial-features-in-sql-server-code-named-denali-community-technology-preview-3.aspx)*.*

**Table of Contents**

* [Spatial Type Improvements](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Spatial_Type_Improvements)
  + [Circular Arcs: Support for Three New Subtypes (enhanced behavior in CTP3 )](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Circular_Arcs_Support_for_Three_New_Subtypes_enhanced_behavior_in_CTP3)
    - [CircularString](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#CircularString)
    - [CompoundCurve](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#CompoundCurve)
    - [CurvePolygon](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#CurvePolygon)
    - [Collections of Curved Objects](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Collections_of_Curved_Objects)
    - [Existing Spatial Methods and Curved Objects](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Existing_Spatial_Methods_and_Curved_Objects)
  + [New and Updated Methods and Aggregates for Both Spatial Types (enhanced behavior in CTP3 )](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#New_and_Updated_Methods_and_Aggregates_for_Both_Spatial_Types_enhanced_behavior_in_CTP3)
  + [Improved Precision](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Improved_Precision)
  + [Geography Type Enhancements](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Geography_Type_Enhancements)
    - [FullGlobe: Support for Geography Objects Larger Than a Logical Hemisphere](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#FullGlobe_Support_for_Geography_Objects_Larger_Than_a_Logical_Hemisphere)
    - [New SRID (new to CTP3 )](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#New_SRID_new_to_CTP3)
    - [SPATIAL\_WINDOW\_MAX\_CELLS: New Spatial Index Hint](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#SPATIAL_WINDOW_MAX_CELLS_New_Spatial_Index_Hint)
    - [Compression for Spatial Indexes](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Compression_for_Spatial_Indexes)
    - [Improved “Create Spatial Index” Build Time for Point Data](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Improved_ldquo_Create_Spatial_Index_rdquo_Build_Time_for_Point_Data)
  + [New Nearest Neighbor Query Plan](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#New_Nearest_Neighbor_Query_Plan)
  + [Other Performance Improvements](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Other_Performance_Improvements)
* [Other Improvements](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Other_Improvements)
  + [Spatial Helper Stored Procedures](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Spatial_Helper_Stored_Procedures)
  + [General Engine Improvements That Affect Spatial Types](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#General_Engine_Improvements_That_Affect_Spatial_Types)
    - [Support for Persisted Computed Columns](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Support_for_Persisted_Computed_Columns)
  + [Client-Side Library Changes](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Client-Side_Library_Changes)
* [Conclusion](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Conclusion)
* [Appendix I – Spatial Methods and Aggregates](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Appendix_I_ndash_Spatial_Methods_and_Aggregates)
* [Appendix II – Spatial Type Classes](http://social.technet.microsoft.com/wiki/contents/articles/4136.aspx#Appendix_II_ndash_Spatial_Type_Classes)

**Introduction**

This document covers the new additions in spatial data support, introduced in Microsoft SQL Server 2008, for the SQL Server Code-Named “Denali” Community Technology Preview 1 (CTP1) and Community Technology Preview 3 (CTP3) releases. These new features are organized into the following major sections:

* “Spatial Type Improvements”
* “Performance Improvements”
* “Other Spatial Improvements”

Note that SQL Azure, starting with the SU3 release in June 2010, supports spatial data at a level of functionality equivalent to SQL Server 2008.

In order to fully digest this document the reader should be familiar with spatial data support in SQL Server 2008. Additionally, sections which include new CTP3-based content are identified in the section heading name by adding: “(enhanced behavior in CTP3)” or “(new CTP3 content)”. Specific new CTP3 content is further identified by adding “(new CTP3 content)” where they occur within the section itself.

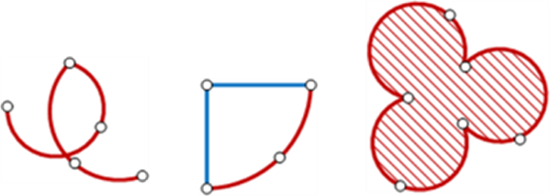
***Note****: The* ***Spatial results*** *tab in SQL Server Management Studio has not been updated to handle the new spatial features present in SQL Server Code-Named* *“Denali” CTP1. Consequently, the* ***Spatial results*** *tab has been disabled and is not currently available in SQL Server Management Studio for the SQL Server Code-Named* *“Denali” CTP1 build. You can, however, use SQL Server Management Studio from an existing installation based on SQL Server 2008 or SQL Server 2008 R2 to connect to an instance of SQL Server Code-Named* *“Denali.” The* ***Spatial results*** *tab is available for spatial query visualization, in this case, but it does not support some of the spatial features that are new with SQL Server Code-Named* *“Denali” CTP1, such as FullGlobe* ***geography*** *type support and circular arcs. Additionally, note that side-by-side installations of SQL Server Code-Named* *“Denali” CTP1 and CTP3 are not supported with existing SQL Server 2008 installations.*

*For SQL Server Code-Named* *“Denali” CTP3, the Spatial results tab is available and handles the new features which SQL Server Code-Named* *“Denali” CTP1 introduced (FullGlobe, circular arcs, etc.) (new CTP3 content).*

**Spatial Type Improvements**

SQL Server Code-Named “Denali” CTP1 introduces several significant enhancements to spatial types, such as support for new subtypes of circular arcs, new and updated methods and aggregates for all subtypes, improved precision, and updates to the geography type.

**Circular Arcs: Support for Three New Subtypes** (enhanced behavior in CTP3)

Circular arcs are new to CTP1 and are based on the ISO SQL/MM, Part 3: Spatial standard. Circular arcs can be defined by themselves or they can be combined with line segments. Circular arcs can also be the basis for a new type of polygon that contains one or more curve components. For example, parallels on the globe (line of latitude) can be expressed as geography circular arcs. CTP3 updates circular arcs with increased precision for the geography type (new CTP3 content). Sample circular arcs are illustrated here.  
  
[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/2526.Figure-1.png)

Circular arcs are supported by **geometry** and **geography** and can be defined using Well-known Text (WKT), Well-known Binary (WKB), and Geography Markup Language (GML) in SQL Server Code-Named “Denali.”

**CircularString**

Circular strings are the basic curve subtype, corresponding to a LineString for linear data. Three points are used to define a segment with the start (first point) and end (third point) points and another point (second point) anywhere along the circular arc. Circular strings can be linked together where the last point of the previous curve becomes the first point of the next curve. Valid circular strings will always have an odd number of points, three or greater.

DECLARE @g GEOGRAPHY;  
SET @g = GEOGRAPHY::STGeomFromText('  
**CIRCULARSTRING(0 -23.43778, 0 0, 0 23.43778)**  
',4326);

**CompoundCurve**

Compound curves enable you to define new curves that can be composed of circular strings only, or circular strings and linear strings. The end point of every component must be identical to the starting point of the next component.

Here is an example of a CompoundCurve made up of circular strings only.

DECLARE @g GEOGRAPHY;  
SET @g = GEOGRAPHY::STGeomFromText('  
**COMPOUNDCURVE(  
CIRCULARSTRING(0 -23.43778, 0 0, 0 23.43778),  
CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),  
CIRCULARSTRING(-90 23.43778, -90 0, -90 -23.43778),  
CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778))**',4326);

Here is an example of a CompoundCurve made up of circular strings and linear strings. Note that linear strings do not contain a keyword.

DECLARE @g GEOGRAPHY;  
SET @g = GEOGRAPHY::STGeomFromText('  
**COMPOUNDCURVE(  
(0 -23.43778, 0 23.43778),** --Linear Segment\* **CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),   
(-90 23.43778, -90 -23.43778),** --Linear Segment\* **CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778))**',4326);

*\*****Note****: You must remove the SQL comments (in green) from this example before it will execute correctly in Transact-SQL.*

**CurvePolygon**

Curve polygons are similar to polygons, having at least one ring and zero or more holes (inner rings). Curve polygons are composed of linear strings, circular strings, and/or compound curves. Within a given ring, the first point as defined in a curve polygon component must be identical to the last point in a curve polygon component, just like standard polygon rings.

Here is an example of a CurvePolygon made up of compound curves, themselves made up of circular strings and linear strings.

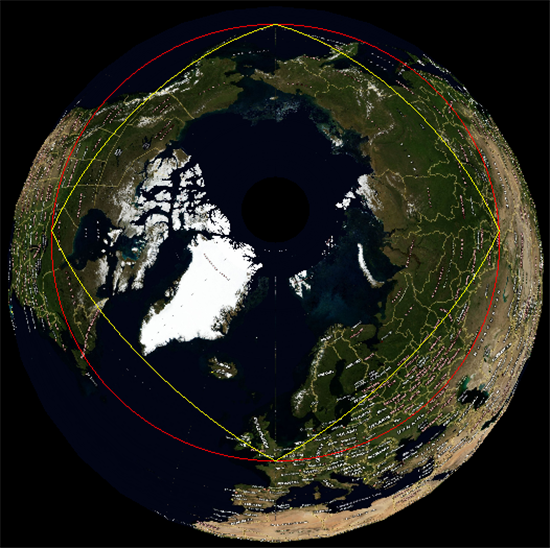
DECLARE @g GEOGRAPHY;  
SET @g = GEOGRAPHY::STGeomFromText('  
**CURVEPOLYGON(  
COMPOUNDCURVE(  
(0 -23.43778, 0 23.43778),  
CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),   
(-90 23.43778, -90 -23.43778),  
CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778)  
)**  
)  
',4326);

**Collections of Curved Objects**

In order to create a collection of curved objects with objects without curves, a geometry collection must be used.

DECLARE @g GEOGRAPHY  
SET @g = GEOGRAPHY::STGeomFromText('  
**GEOMETRYCOLLECTION(  
COMPOUNDCURVE(  
CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),  
(-90 23.43778, -90 -23.43778)   
),   
COMPOUNDCURVE(   
CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778),   
(0 -23.43778, 0 23.43778)   
)**  
**)**',4326);

The following illustration shows the difference between circular arcs and linear strings for the same set of coordinates.

[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/6874.Figure-2.png)

CIRCULARSTRING(0 50, 90 50, 180 50, 270 50, 0 50) -- RED LINE  
and  
LINESTRING (0 50, 90 50, 180 50, 270 50, 0 50) –- YELLOW LINE

***Note****: The circular string defined in the previous example represents a circle on the globe, defined by five points in which the first and last coordinates are the same. You can create this feature as a curve polygon as follows.*

DECLARE @g GEOGRAPHY  
SET @g = GEOGRAPHY::STGeomFromText('  
**CURVEPOLYGON(  
CIRCULARSTRING(0 50, 90 50, 180 50, 270 50, 0 50)  
)**',4326);

**Existing Spatial Methods and Curved Objects**

All existing methods work on circular objects (STIntersects is illustrated here).  
  
[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/4034.Figure-3.png)  
  
New Methods for Circular Arcs (enhanced behavior in CTP3) BufferWithCurves() uses circular arcs to construct buffered object with greatly reduced number of points compared to STBuffer().  
[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/3036.Figure-4.png)  
In CTP3, BufferWithCurves()of a point object constructs a curve polygon consisting of only 2 arcs, compared to 3 arcs in CTP1 (enhanced behavior in CTP3).

**STNumCurves()** and **STCurveN()** are used for iteration through the list of the circular arc edges.

**STCurveToLine()** and **CurveToLineWithTolerance()** are used for approximating circular arcs with line segments within default and user specific tolerance.

***Note****: STCurveToLine and CurveToLineWithTolerance can be used to convert both LineStrings/MultiLineStrings and Polygons/MultiPolygons that have circular arc components to linear representations of the same dimensionality. This can use useful for:*

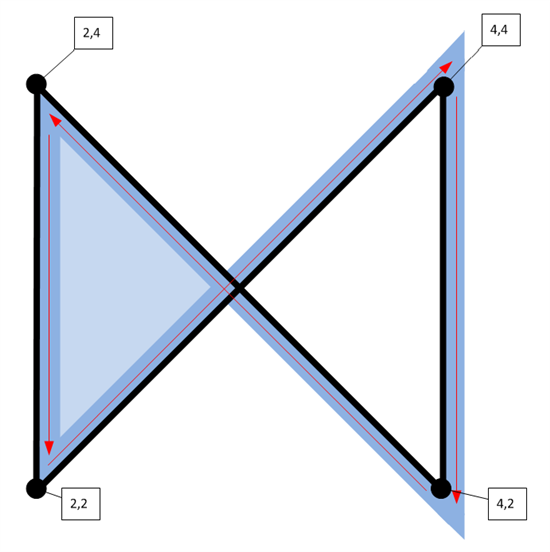
* *Spatial object representation in visualization programs*
* *Backward compatibility with earlier versions of SQL Server that have spatial data support*

**New and Updated Methods and Aggregates for Both Spatial Types** (enhanced behavior in CTP3)

**IsValidDetailed()** returns a message that can help in identifying validity issues with a spatial object. Only the first error found is returned unless the object is valid, in which case a value of 24400 is returned (new to CTP3).

Messages:

* 24400: Valid
* 24401: Not valid, reason unknown.
* 24402: Not valid because point ({0}) is an isolated point, which is not valid in this type of object.
* 24403: Not valid because some pair of polygon edges overlap.
* 24404: Not valid because polygon ring ({0}) intersects itself or some other ring.
* 24405: Not valid because some polygon ring intersects itself or some other ring.
* 24406: Not valid because curve ({0}) degenerates to a point.
* 24407: Not valid because polygon ring ({0}) collapses to a line at point ({1}).
* 24408: Not valid because polygon ring ({0}) is not closed.
* 24409: Not valid because some portion of polygon ring ({0}) lies in the interior of a polygon.
* 24410: Not valid because ring ({0}) is the first ring in a polygon of which it is not the exterior ring.
* 24411: Not valid because ring ({0}) lies outside the exterior ring ({1}) of its polygon.
* 24412: Not valid because the interior of a polygon with rings ({0}) and ({1}) is not connected.
* 24413: Not valid because of two overlapping edges in curve ({0}).
* 24414: Not valid because an edge of curve ({0}) overlaps an edge of curve ({1}).
* 24415: Not valid some polygon has an invalid ring structure.
* 24416: Not valid because in curve ({0}) the edge that starts at point ({1}) is either a line or a degenerate arc with antipodal endpoints.

The following example of an invalid spatial object illustrates how the new IsValidDetailed() methods behaves.  
[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/2541.Figure-5.png)  
DECLARE @p GEOGRAPHY = 'Polygon((2 2, 4 4, 4 2, 2 4, 2 2))'  
SELECT @p.**IsValidDetailed()**--Returns: 24409: Not valid because some portion of polygon ring (1) lies in the interior   
--of a polygon.

**HasZ** returns 1 if spatial objects contains at least one Z value, otherwise 0 is returned (new to CTP3).

DECLARE @p GEOGRAPHY = 'Point(1 1 1 1)'  
SELECT @p.**HasZ**   
--Returns: 1 (true)

**HasM** returns 1 if spatial objects contains at least one M value, otherwise 0 is returned (new to CTP3).

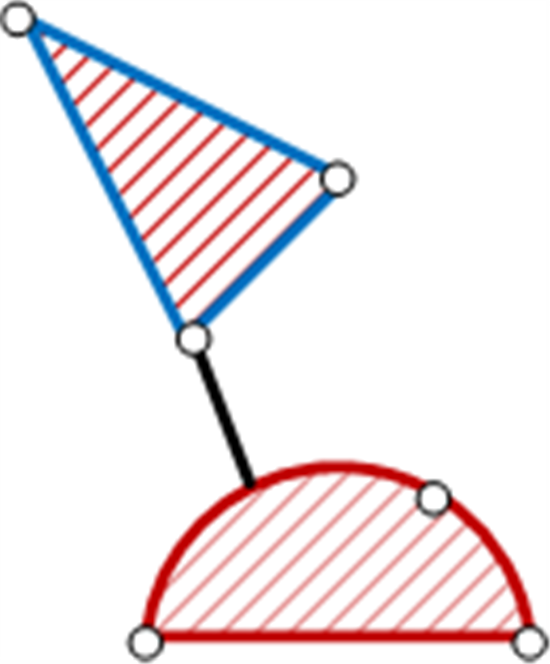
DECLARE @p GEOGRAPHY = 'Point(1 1 1 1)'  
SELECT @p.**HasM**   
--Returns: 1 (true)

**AsBinaryZM()**adds support for Z and M values to the OGC WKB format (new to CTP3).

DECLARE @g1 GEOMETRY = 'Point(1 1 2 3)';  
SELECT @g1.STAsBinary();  
-- Returns: 0x0101000000000000000000F03F000000000000F03F

SELECT @g1.**AsBinaryZM()**;  
--Returns: 0x01B90B0000000000000000F03F000000000000F03F00000000000000400000000000000840

**ShortestLineTo()** returns a LineString that represents the shortest distance between two objects. The resulting LineString does not have to touch either parent object on a shape-defining vertex point.

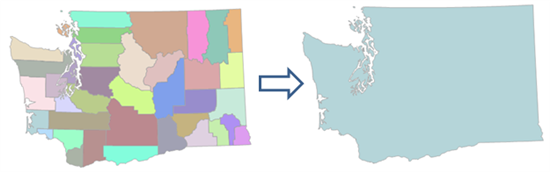
[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/0488.Figure-6.png)  
  
The following example is not related to the previous graphic:

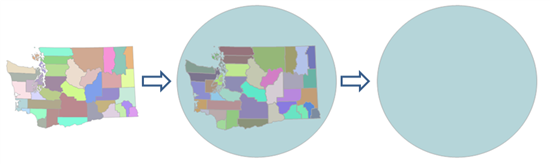
DECLARE @a GEOGRAPHY = 'LineSTring(-118 34, -119 35)'  
DECLARE @b GEOGRAPHY = 'Polygon((-120 22, -119 22, -119 23, -120 23, -120 22))'  
SELECT @a.**ShortestLineTo(@b)**.STAsText()  
--Returns: LINESTRING (-118.00000000000001 34.000000000000007, -119 23.000000000000004)

The followingaggregates are now available. They are exposed as static methods on **geography** and **geometry** types:

* UnionAggregate
* EnvelopeAggregate
* CollectionAggregate
* ConvexHullAggregate

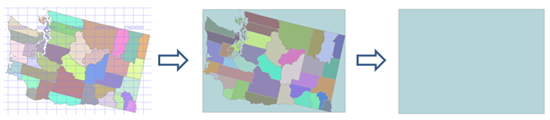
***Note****: The new aggregates are exposed in SQL Server only and are not exposed in the underlying spatial library.*

**UnionAggregate()** combines multiple spatial objects into a single spatial object, removing interior boundaries, where applicable.  
  
SELECT **Geography::UnionAggregate(geog)** FROM Counties  
WHERE name = 'Washington';  
  
[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/3487.Figure-7.PNG)

**EnvelopeAggregate()** for the **geography** type returns a bounding circular object as a CurvePolygon which encloses one or more spatial objects.  
  
SELECT **Geography::EnvelopeAggregate(geog)** FROM Counties  
WHERE name = 'Washington';  
  
[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/0160.Figure-8.PNG)

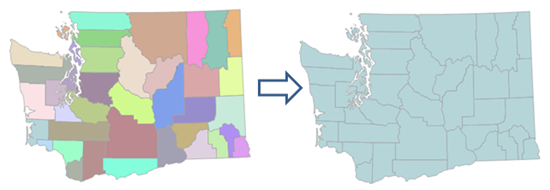
***Note****: Geography::EnvelopeAggregate() returns a CurvePolygon. To convert this object to a “traditional” polygon, use the STCurveToLine() method.*

**EnvelopeAggregate()** for the **geometry** type returns a 5 point polygon (rectangle) which encloses one or more spatial objects.

SELECT **Geometry::EnvelopeAggregate(geom)** FROM Counties  
WHERE name = 'Washington';  
 *[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/6840.Figure-9.PNG)*

**CollectionAggregate()** returns a geometry collection with one geometry part for each spatial object(s) in the selection set.

SELECT **Geography::CollectionAggregate(geog)** FROM Counties  
WHERE name = 'Washington';

[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/0243.Figure-10.PNG)

**ConvexHullAggregate()** returns a convex hull polygon which encloses one or more spatial objects  
  
SELECT **Geography::ConvexHullAggregate(geog)** FROM Counties  
WHERE name = 'Washington'

|  |
| --- |
| **STLength()** has been updated an now works on both valid and invalid LineStrings. This was done for LineStrings because the MakeValid method can remove overlapping parts. Typically a LineString is invalid due to overlapping segments, which may be caused by anomalies such as inaccurate GPS traces.  ***Note****: STLength() does not remove overlapping/invalid segments. Additionally, STLength() includes overlapping and invalid segments in the length value it returns.*  **MinDbCompatibilityLevel()** is a new method is used for backward compatibility. It indicates whether spatial objects can be recognized by SQL Server 2008 and SQL Server 2008 R2.  DECLARE @C GEOMETRY SET @C = GEOMETRY::Parse('CIRCULARSTRING(0 50, 90 50, 180 50)')  SELECT @C.**MinDbCompatibilityLevel()** --Result: 110  DECLARE @L GEOMETRY SET @L = GEOMETRY::Parse('LINESTRING (0 50, 90 50, 180 50)') SELECT @L.**MinDbCompatibilityLevel()** --Result: 100  ***Note****: You can set the compatibility level of the database with the following Transact-SQL statement.*  --Set database compatibility level to SQL Server Code-Named "Denali" **ALTER DATABASE *database***  **SET COMPATIBILITY\_LEVEL = 110**;  --Set database compatibility level to SQL Server 2008 **ALTER DATABASE *database*** **SET COMPATIBILITY\_LEVEL = 100**;  **Improved Precision**  All constructions and relations are now done with 48 bits of precision, compared to 27 bits used in SQL Server 2008 and SQL Server 2008 R2. This can reduce the error caused by rounding of floating point coordinates for original vertex points by the internal computation.  For example, consider the following coordinate, which was processed using the STUnion() method in SQL Server 2008 but which was not involved in the resulting geometry.  Original Vertex Coordinate Original Vertex Coordinate After Computation  82.339026 29.661245 82.339025999885052 29.662144999951124  In SQL Server Code-Named “Denali,” the greater numerical precision assists in the preservation of original coordinates of input points (vertex), in most cases. Here is the result of the same STUnion() method in SQL Server Code-Named “Denali,” shown earlier.  Original Vertex Coordinate Original Vertex Coordinate After Computation  82.339026 29.661245 82.339026 29.662145  Sample code illustrating the improved precision in SQL Server Code-Named “Denali” compared to SQL Server 2008 (SQL Server 2008 R2):  DECLARE @g1 GEOMETRY = 'POLYGON(( 0 0, 82.339026 0,  82.339026 29.661245,  0 29.661245,  0 0))'  DECLARE @g2 GEOMETRY = 'POINT(0 0)' SELECT @g1.STUnion(@g2).ToString() --RESULTS:  --  --From SQL Server “Denali” Server Instance  --POLYGON ((0 0,  -- 82.339026 0,  -- 82.339026 29.661245,  -- 0 29.661245,  -- 0 0))  --  --From SQL Server 2008/R2 Instance  --POLYGON ((0 0,  -- 82.339025497436523 0,  -- 82.339025497436523 29.661244869232178,  -- 0 29.661244869232178,  -- 0 0))  Note that the highlighted coordinate did not change its apparent precision after the union operation when executed on the SQL Server Code-Named “Denali” instance, illustrating the effect of the improved precision in the new server.  **Geography Type Enhancements**  **FullGlobe: Support for Geography Objects Larger Than a Logical Hemisphere**  The SQL Server Code-Named “Denali” Spatial Library supports objects that are larger than a logical hemisphere. Restricted to slightly less than a logical hemisphere in SQL Server 2008, geography featuresin SQL Server Code-Named “Denali” can now be as big as entire globe. A new type of object, called FULLGLOBE, can be constructed or received as a result of an operation. You can also construct objects that are “small holes” (see the following picture), because the interior for **geography** objects is defined by the orientation of its rings. There is no difference between exterior and interior rings in the **geography** data type.    [The first image shows an example of a “small hole” in a FullGlobe object, the second image shows a regular polygon, and the third image shows the union of the two, yielding a spatial object that covers the Earth with three small holes.](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)  [Here is a simple example that constructs a new FullGlobe object and executes a method on that object.](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)  [DECLARE @g GEOGRAPHY = GEOGRAPHY::STGeomFromText('](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)**[FULLGLOBE](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)**[',4326); SELECT @g.STArea()-- calculate the area of the WGS84 ellipsoid --Result: 510,065,621,710,996 meters squared](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)  [Vertex order is critical for the](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG) **[geography](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)** [type. In SQL Server 2008 and SQL Server 2008 R2, if you enter an “incorrect” coordinate order for a geography polygon, you receive an error:](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)  [The specified input does not represent a valid geography instance because it exceeds a single hemisphere. Each geography instance must fit inside a single hemisphere. A common reason for this error is that a polygon has the wrong ring orientation.](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)  [For the SQL Server](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG) **[geography](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)** [data type, ring order is defined by the](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG) *[left-foot rule](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)*[. The left-foot rule specifies the interior region of the polygon (when you “walk” the boundary of a polygon, your left foot is always inside). Traditional outer ring/inner ring (hole) relationships can be modeled on the closed surface of the globe by using this definition.](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)  ***[Note](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)****[: The term “left-foot rule” is used in deference to the term “left-hand rule,” which is already in use by physicists and mathematicians to describe phenomena other than polygon ring order.](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)*  [The following examples demonstrate how the](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG) **[geography](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)** [type deals with large (greater than a logical hemisphere) objects.  Here is an example of a small (less than a logical hemisphere) polygon.](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)  [DECLARE @R GEOGRAPHY;  SET @R = GEOGRAPHY::Parse('Polygon((-10 -10, 10 -10, 10 10, -10 10, -10 -10))'); SELECT @R;  The resulting object can be visualized as follows.](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/1401.Figure-12.PNG)    The following Transact-SQL changes the coordinate order of the polygon ring to the opposite direction of the previous example.  DECLARE @R GEOGRAPHY;  SET @R = GEOGRAPHY::Parse('Polygon((-10 -10, -10 10, 10 10, 10 -10, -10 -10))'); SELECT @R;  The resulting object is now the “rest of the globe.”  Other Considerations for the geography type  There are several other considerations which need to be taken into account with the FullGlobe enhancements to the **geography** type:   * FullGlobe cannot be a member of a GeometryCollection * WKT, WKB and GML (and its schema) now support FullGlobe objects * Antipodal edges are not allowed (for example, Linestring(0 90, 0 -90) is considered antipodal while LineString(0 90, 0 0, 0 -90) is not) * There are now two types of arcs in **geography**: great circle arcs defined with two points and small circle arcs defined with three points * Degenerate arcs will be treated as great circle arcs between the start and end points for the **geography** type (for the **geometry** type such arcs will be treated as straight edges) * EnvelopeAngle() will return 180 for objects larger than a logical hemisphere and < 90 for smaller than a logical hemisphere objects |

New Methods for the geography Type

The **geography** type now allows invalid objects to be inserted into a table. The following new methods deal with such objects.

**STIsValid()** and **MakeValid()** allow invalid **geography** objects to be detected and corrected in a similar fashion to the **geometry** type.

**Geography** polygons can now be accommodated without regard to ring orientation. This can lead to unintended behavior. The following new method allows such polygons to be reoriented.

**ReorientObject()**can be used to reorient polygon rings for cases where they are constructed with the wrong orientation. LineStrings, curves and points will not be touched.

Here is an example.

DECLARE @R GEOGRAPHY = GEOGRAPHY::Parse('Polygon((-10 -10, -10 10, 10 10,   
10 -10, -10 -10))');  
SELECT @R.**ReorientObject()**.STAsText();  
--Result: POLYGON ((10 10, -10 10, -10 -10, 10 -10, 10 10))

**STWithin()**, **STContains()**, **STOverlaps()**, and **STConvexHull()** methods have been added for the **geography** type, previously available only for the **geometry** type. With the exception of STConvexHull(), these new methods are supported by spatial indexes.

|  |
| --- |
| **New SRID** (new to CTP3)  A new spatial reference id (SRID), 104001, has been added to the sys.spatial\_reference\_systems view. This new SRID is an Earth unit sphere (a sphere of radius 1) and can be used with the geography type to perform optimized numerical computations when more precise ellipsoidal mathematics are not required. Furthermore, Distance(), Length() and ShortestLineTo() methods are optimized to run faster on the unit sphere than their counterparts on the ellipsoid. To compute measures, such as length and area, on a given sphere (as defined by its radius), simply multiply results by the radius of the desired sphere:   * For linear measure (length, distance, buffer distance): length \* (sphere radius) * For area measure: area \* (sphere radius) \* (sphere radius)   The following examples illustrate how to work with unit sphere on the geography type.  -- LineString length along the Prime Meridian from the Equator to the -- North Pole on the unit sphere… SELECT GEOGRAPHY::STGeomFromText('Linestring(0 0,90 0)',**104001**).STLength() --Returns: 1.5707963267949 (PI/2)  -- LineString length on the WGS84 Sphere (major radius = 6378137 meters)… DECLARE @r int = 6378137; SELECT GEOGRAPHY::STGeomFromText('Linestring(0 0,90 0)',**104001**).STLength()\*@r --Returns: 10018754.1713946 meters  -- Area of the FULLGLOBE on the unit sphere… SELECT GEOGRAPHY::STGeomFromText('Fullglobe',**104001**).STArea() --Returns: 12.5663706143592 as the area of the unit sphere  -- To return the area of the WGS84 Sphere (major radius =  -- 6378137 meters) multiple the area time the sphere radius squared: DECLARE @r bigint = 6378137; SELECT GEOGRAPHY::STGeomFromText('Fullglobe',**104001**).STArea()\*(@r\*@r) --Returns: 511207893395811 as the area of the WGS84 sphere in square meters |

**Performance Improvements** (enhanced behavior in CTP3)

SQL Server Code-Named “Denali” CTP1 includes features and enhancements that improve performance. These include new spatial indexes, a spatial index hint, compression, improved index build time, and a new query plan. Additionally, CTP3 provides further performance enhancements at the spatial library level for Distance() and ShortestLineTo() methods with corresponding performance improvements at the SQL level for methods which are based on them including STDistance() and ShortestLineTo() (new CTP3 content).

Spatial Index Improvements

New spatial indexes provide further levels of tuning, and a new hint helps you fine-tune query performance.

New Spatial Indexes: Geometry Auto Grid and Geography Auto Grid

A new auto grid spatial index is available for both spatial types (geometry\_auto\_grid and geography\_auto\_grid). The GRIDS parameter is no longer valid or needed when this index type is used. The new auto grid actually uses a different strategy to pick the right tradeoff between performance and efficiency. It uses eight levels (instead of four) for better approximation of objects of various sizes. The older style spatial index with four user-specified levels is still supported and is now referred to as a “manual grid” spatial index.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The new geography auto grid also eliminates the performance penalty on queries using a geography manual grid index when the indexed geography column contains EMPTY or NULL spatial data.  Here are some examples that use the new auto grid for the **geography** data type.  CREATE SPATIAL INDEX idxGeog ON *table*(*column*) USING **GEOGRAPHY\_AUTO\_GRID**;  or  CREATE SPATIAL INDEX idxGeog ON *table*(*column*) USING **GEOGRAPHY\_AUTO\_GRID** WITH (CELLS\_PER\_OBJECT = 32);  If an index type is not specified in the CREATE SPATIAL INDEX DDL, the appropriate auto grid for the specified spatial column data type is used by default. In the following example, if the spatial column data type is **geography**, the GEOGRAPHY\_AUTO\_GRID is used by default.  CREATE SPATIAL INDEX idxGeog  ON *table*(*column*);  The following figure illustrates spatial object approximation with 64, 128, and 256 cells\_per\_object, respectively, on the new eight-level auto grid. New auto grid spatial indexes have better continuous coverage and higher precision.  The default number of cells per object for the new auto grid is 8 for **geometry** and 12 for **geography**. For original GEOMETRY\_GRID and GEOGRAPHY\_GRID indexes, the default for CELLS\_PER\_OBJECT remains the same as previous at 16.  ***Note****: The CELLS\_PER\_OBJECT parameter is not used for point objects. It is used for all other objects, including multipoint objects.*  **SPATIAL\_WINDOW\_MAX\_CELLS: New Spatial Index Hint**  This new spatial hint is critical for fine-tuning query performance using a spatial index. Dense spatial data often requires a higher SPATIAL\_WINDOW\_MAX\_CELLS, whereas sparse spatial data often demands a lower SPATIAL\_WINDOW\_MAX\_CELLS for optimum performance.  This hint can be used as follows.  SELECT \* FROM table t with(**SPATIAL\_WINDOW\_MAX\_CELLS=1024**) WHERE t.geom.STIntersects(@window)=1;  SPATIAL\_WINDOW\_MAX\_CELLS hint does not guarantee that spatial index would be used in the query plan. However, if it is used hint will override the default WINDOW\_MAX\_CELLS parameter.  The default WINDOW\_MAX\_CELLS parameter is 512 for **geometry** and 768 for **geography**. Higher values make the spatial index more efficient, but the primary filter is slower. Lower values make the primary filter faster, but more time is spent in the secondary filter due to potentially poorer index efficiency, with the optimum somewhere in between. For example, in the following illustration, 256 cells is the optimum query window.    This chart shows query execution time (in milliseconds) compared to the old configurations (HHHH, HHMM, and MMMM in the three left-hand columns). Total times are combined primary filter times and secondary filter times. Making spatial indexes more precise improves overall performance until the tipping point is reached.  ***Note****: For SQL Server 2008, this parameter was always set to 1024 and was not modifiable.*  **Compression for Spatial Indexes**  Compression can be enabled on spatial indexes with Transact-SQL DDL.  CREATE SPATIAL INDEX idxGeom ON *table*(*column*) USING GEOGRAPHY\_GRID WITH ( **DATA\_COMPRESSION = *page|row*** );  On the basis of internal tests, spatial indexes with compression are 40-50 percent smaller, with a 5-10 percent performance overhead.  **Improved “Create Spatial Index” Build Time for Point Data**  Spatial index build time for point data can be as much as four to five times faster than it is under similar conditions in SQL Server 2008.  **New Nearest Neighbor Query Plan**  A nearest neighbor query plan can be expressed as follows (the query elements highlighted in **red** are required for making the query optimizer process queries using a nearest neighbor query plan).  SELECT **TOP**(5) \* FROM Restaurants r WHERE r.type = 'Italian'  AND r.position.**STDistance(**@location**) IS NOT NULL** **ORDER BY** r.position.**STDistance**(@location);  It can also be expressed with limited maximum distance via the @max\_range variable, below.  SELECT **TOP**(5) \* FROM Restaurants r WHERE r.type = 'Italian' AND r.position.**STDistance(**@location**) < @max\_range** **ORDER BY** r.position.**STDistance**(@location);  This enhancement uses a spatial index to filter out rows and speed up query execution. With a spatial index, nearest neighbor execution time is up to two orders of magnitude faster (10 to 30 times) than a counterpart query in SQL Server 2008.  **Other Performance Improvements**  The methods STIntersects(), STWithin(), STContains(), and STOverlaps() are now optimized when at least one of the operands is a point.  STDistance() performance between two points has been optimized.  STBuffer() has been optimized and improved for all cases. It’s faster and has lower memory footprint, especially for large and complex objects.  **Other Improvements**  SQL Server Code-Named “Denali” also introduces other improvements to the way spatial data is handled. These improvements include two new stored procedures, general engine improvements, and client-side library changes.  **Spatial Helper Stored Procedures**  Two new helper stored procedures are available:   * **sp\_help\_spatial\_geography\_histogram** * **sp\_help\_spatial\_geometry\_histogram**   These routines can be used to evaluate the distribution of spatial data within a table over a given spatial column.  For example, the following query uses the **sp\_help\_spatial\_geography\_histogram** stored procedure to return information about the distribution of spatial data in the Mexico table for the geography column.  EXEC sp\_help\_spatial\_geography\_histogram Mexico,geography,1000,100  Arguments: <table name>, <spatial column>, <cell size>, <sample %>  Here is sample output from the previous query.    The output can be used for investigating spatial index efficiency or analyzing spatial data in general. The histograms can be also shown in the SQL Server Management Studio **Spatial results** tab (greater detail is shown in the image to the right).  With some additional programming, data from these stored procedures can be used for external visualization, shown over Bing Maps.  **General Engine Improvements That Affect Spatial Types**  SQL Server Code-Named “Denali” CTP1 includes support for persisted computed columns and improved CLR UDT performance.  **Support for Persisted Computed Columns**  UDTs and spatial types can now be persisted in computed columns as shown in the following example.  CREATE TABLE location ( id int primary key,  x float(max),  y float(max),  range float(max),  **geom as**  **geometry::Point(x, y, 0)** **.STBuffer(range) PERSISTED)**;  **Improved CLR UDT Aggregate Performance**  CLR UDT aggregates in general (and by inference, new spatial aggregates) are improved and have much better performance. They are now 5 to 20 times faster compared to SQL Server 2008.  **Client-Side Library Changes**  New sink interfaces, IGeometrySink110 and IGeographySink110, are available, and you should plan to use them in your future programming. They must be used for the following types of objects, which are incompatible with SQL Server 2008:   * Objects bigger than a logical hemisphere * Objects with circular arcs   However, the old sinks will continue to work for SQL Server 2008 compatible objects.  Geometry and Geography builders (SqlGeometryBuilder and SqlGeographyBuilder) now support circular arc constructions.  A new method, Deserialize, has been added on the client library to both types. This method simplifies deserialization. The following code shows how to use it to deserialize SqlBytes.  using (SqlConnection con = new SqlConnection("...")) { con.Open(); SqlCommand cmd = new SqlCommand("...", con); SqlDataReader reader = cmd.ExecuteReader(); while (reader.Read()) { SqlGeography g = SqlGeography.**Deserialize**(reader.GetSqlBytes(0)); Console.WriteLine(g) } }  **Conclusion**  The new spatial features in SQL Server Code-Named “Denali” CTP1 and CTP3 represent a significant milestone in the evolution of SQL Server spatial data support. The ability to support full globe spatial objects and circular arcs on the ellipsoid are industry firsts for relational database systems. The **geography** data type has achieved parity relative to the function and variety of methods that support the **geometry** data type. Overall performance, from spatial indexes to methods, has been significantly increased. These and other improvements to spatial promise a significant step forward in spatial prowess of the next generation of SQL Server.  **Appendix I – Spatial Methods and Aggregates**  Entries in black type indicate methods/aggregates which were present in SQL Server 2008 and SQL Server 2008 R2. Green and blue cells indicate methods/aggregates which are new to SQL Server Code-named “Denali”. Where these occur in table, below, the third column, Denali CTP, indicates in which Denali Community Technology Preview they first appeared in for SQL Server Code-Named “Denali”. Blank entries indicate no current method/aggregate. Entries with “*na*” indicate not applicable.  There are 83 Geography and 89 Geometry methods.   |  |  |  |  | | --- | --- | --- | --- | | **GEOGRAPHY** | **GEOMETRY** | **Denali CTP** | **Minimum DB Compatibility Level** | | **OGC Methods** | | | | | STArea() | STArea() |  | 100 | | STAsBinary() | STAsBinary() |  | 100 | | STAsText() | STAsText() |  | 100 | |  | STBoundary() |  | 100 | | STBuffer() | STBuffer() |  | 100 | | *see EnvelopeCenter()* | STCentroid() |  | 100 | | STContains() | STContains() | CTP1 | 100 | | STConvexHull() | STConvexHull() | CTP1 | 100 | |  | STCrosses() |  | 100 | | STCurveN() | STCurveN() | CTP1 | 110 | | STCurveToLine() | STCurveToLine() | CTP1 | 110 | | STDifference() | STDifference() |  | 100 | | STDimension() | STDimension() |  | 100 | | STDisjoint() | STDisjoint() |  | 100 | | STDistance() | STDistance() |  | 100 | | STEndPoint() | STEndPoint() |  | 100 | | *see EnvelopeAggregate()* | STEnvelope() |  | 100 | | STEquals() | STEquals() |  | 100 | | *na* | STExteriorRing() |  | 100 | | STGeometryN() | STGeometryN() |  | 100 | | STGeometryType() | STGeometryType() |  | 100 | | *na* | STInteriorRingN() |  | 100 | | STIntersections() | STIntersections() |  | 100 | | STIntersects() | STIntersects() |  | 100 | | STIsClosed() | STIsClosed() |  | 100 | | STIsEmpty() | STIsEmpty() |  | 100 | | *na* | STIsRing() |  | 100 | |  | STIsSimple() |  | 100 | | STIsValid() | STIsValid() | CTP1 | 100 | | STLength() | STLength() |  | 100 | | STNumCurves() | STNumCurves() | CTP1 | 110 | | STNumGeometries() | STNumGeometries() |  | 100 | | *na* | STNumInteriorRing() |  | 100 | | STNumPoints() | STNumPoints() |  | 100 | | STOverlaps() | STOverlaps() | CTP1 | 100 | | STPointN() | STPointN() |  | 100 | |  | STPointOnSurface() |  | 100 | |  | STRelate() |  | 100 | | STSrid | STSrid |  | 100 | | STStartPoint() | STStartPoint() |  | 100 | | STSymDifference() | STSymDifference() |  | 100 | |  | STTouches() |  | 100 | | STUnion() | STUnion() |  | 100 | | STWithin() | STWithin() | CTP1 | 100 | | *na* | STX |  | 100 | | *na* | STY |  | 100 | | **Extended Methods** | | |  | | AsBinaryZM() | AsBinaryZM() | CTP3 | 110 | | AsGML() | AsGML() |  | 100 | | ASTextZM() | ASTextZM() |  | 100 | | BufferWithCurves() | BufferWithCurves() | CTP1 | 110 | | BufferWithTolerance() | BufferWithTolerance() |  | 100 | | CurveToLineWithTolerance() | CurveToLineWithTolerance() | CTP1 | 110 | | EnvelopeAngle() | *na* |  | 100 | | EnvelopeCenter() | *na* |  | 100 | | Filter() | Filter() |  | 100 | | HasM | HasM | CTP3 | 100 | | HasZ | HasZ | CTP3 | 100 | | InstanceOf() | InstanceOf() |  | 100 | | IsNull() | IsNull() |  | 100 | | IsValidDetailed() | IsValidDetailed() | CTP3 | 100 | | Lat | *na* |  | 100 | | Long | *na* |  | 100 | | M | M |  | 100 | | MakeValid() | MakeValid() | CTP1 | 100 | | MinDbCompatibilityLevel() | MinDbCompatibilityLevel() | CTP1 | 100 | | Reduce() | Reduce() |  | 100 | | ReorientObject() | *na* | CTP1 | 100 | | NumRings() | *na* |  | 100 | | RingN() | *na* |  | 100 | | ShortestLineTo() | ShortestLineTo() | CTP1 | 100 | | ToString() | ToString() |  | 100 | | Z | Z |  | 100 | | **OGC Static Methods** | | |  | | STGeomCollFromText() | STGeomCollFromText() |  | 100 | | STGeomFromText() | STGeomFromText() |  | 100 | | STLineFromText() | STLineFromText() |  | 100 | | STPointFromText() | STPointFromText() |  | 100 | | STPolyFromText() | STPolyFromText() |  | 100 | | STMLineFromText() | STMLineFromText() |  | 100 | | STMPointFromText() | STMPointFromText() |  | 100 | | STMPolyFromText() | STMPolyFromText() |  | 100 | | STGeomCollFromWKB() | STGeomCollFromWKB() |  | 100 | | STGeomFromWKB() | STGeomFromWKB() |  | 100 | | STLineFromWKB() | STLineFromWKB() |  | 100 | | STPointFromWKB() | STPointFromWKB() |  | 100 | | STPolyFromWKB() | STPolyFromWKB() |  | 100 | | STMLineFromWKB() | STMLineFromWKB() |  | 100 | | STMPointFromWKB() | STMPointFromWKB() |  | 100 | | STMPolyFromWKB() | STMPolyFromWKB() |  | 100 | | **Extended Static Methods** | | |  | | GeomFromGML() | GeomFromGML() |  | 100 | | Parse() | Parse() |  | 100 | | Point() | Point() |  | 100 | | Null() | Null() |  | 100 | | **Aggregates (Extended Static Methods)** | | |  | | CollectionAggregate() | CollectionAggregate() | CTP1 | 100 | | ConvexHullAggregate() | ConvexHullAggregate() | CTP1 | 100 | | EnvelopeAggregate() | EnvelopeAggregate() | CTP1 | 100 | | UnionAggregate() | UnionAggregate() | CTP1 | 100 |   *Note that all methods prefixed with “ST” are OGC-compatible.*  **Appendix II – Spatial Type Classes**  Entries in black type indicate classes which were present in SQL Server 2008 and SQL Server 2008 R2. Red type indicates classes which are new to SQL Server Code-Named “Denali”. Where these occur in table, below, the third column, Denali CTP, indicates in which Denali Community Technology Preview they first appeared. Entries with “*na*” indicate not applicable.   |  |  |  |  | | --- | --- | --- | --- | | **Geography** | **Geometry** | **Denali CTP** | **Minimum DB Compatibility**  **Level** | | Point | Point |  | 100 | | LineString | LineString |  | 100 | | Polygon | Polygon |  | 100 | | MultiPoint | MultiPoint |  | 100 | | MultiLineString | MultiLineString |  | 100 | | MultiPolygon | MultiPolygon |  | 100 | | GeometryCollection | GeometryCollection |  | 100 | | FullGlobe | *na* | CTP1 | 110 | | CircularString | CircularString | CTP1 | 110 | | CompoundCurve | CompoundCurve | CTP1 | 110 | | CurvePolygon | CurvePolygon | CTP1 | 110 |   *Note that all classes are OGC-compatible with the exception of FullGlobe, which is a new concept.* ***Note****: you may download this document as a Microsoft Word file from* [*sqlcat.com [http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-components-sitefiles/10_5F00_external.png](http://sqlcat.com/whitepapers/archive/2011/08/08/new-spatial-features-in-sql-server-code-named-denali-community-technology-preview-3.aspx)*](http://sqlcat.com/whitepapers/archive/2011/08/08/new-spatial-features-in-sql-server-code-named-denali-community-technology-preview-3.aspx)*.* |